Equalizator
Détection de genre musical par feature-extraction

13 avril 2016

Table des matières

1 Introduction **2**

2 Feature extraction **2**
2.1 Idée . . . 2
2.2 Implémentation . . . 2
2.3 Résultats . . . 4

3 Régression logistique **4**
3.1 Principe . . . 4
3.2 Exploitation . . . 4
3.3 Résultats . . . 4

4 Annexes **5**
4.1 Programme d'analyse . . . 5
4.2 Regression logistique . . . 7
4.3 Fonction de prédiction . . . 7

1 Introduction

L'une des principales fonctionnalités de notre produit est de pouvoir détecter automatiquement le genre musical d'une chanson et d'ainsi adapter son égalisation.

Nous avons commencé la réalisation d'un algorithme de détection de genre musical basé sur la *feature extraction* et une régression, des approches typiques du machine learning.

2 Feature extraction

2.1 Idée

Notre travail est basé sur un article publié en 2001 par l'université de Princeton.

Il s'agit d'analyser le spectre audio des morceaux de musique afin d'en extraire certaines propriétés. Nous avons implémenté le calcul de 5 de ces propriétés, qui sont :

- Le *Mean-Centroid* mesure de la brillance du spectre audio. Une musique comprenant beaucoup d'aigus et/ou beaucoup d'instruments aura tendance à avoir un *Mean-Centroid* plus élevé
- Le *Mean-Rolloff* mesure de la répartition du spectre audio. Plus un morceaux contient de basses, plus ce nombre sera grand.
- Le *Mean-Flux* mesure le "mouvement" dans le spectre audio. Si celui-ci varie rapidement et fréquemment, ce nombre est plus grand.
- Le *ZeroCross* mesure la quantité de "bruit" d'un morceaux. Un morceau calme aura un *ZeroCross* plus faible.
- Le *LowEnergy* mesure la quantité de passages dont l'énergie est faible par rapport à la moyenne. Un morceau agité aura un *LowEnergy* faible.

On peut ainsi obtenir 5 nombres pour chaque chanson et les classifier en fonction de ceux-ci.

2.2 Implémentation

Nous avons utilisé le logiciel libre Scilab. Le code peut être trouvé en Annexe. Nous avons principalement eu recours à la Transformée de Fourier Discrète pour cet algorithme.

Nous avons très rapidement vu de grandes différences dans les propriétés que nous avons extraites de différents morceaux. Les graphiques suivants proviennent des morceaux "Pixel Party" de Vexento et du "Printemps" des 4 saisons de Vivaldi.

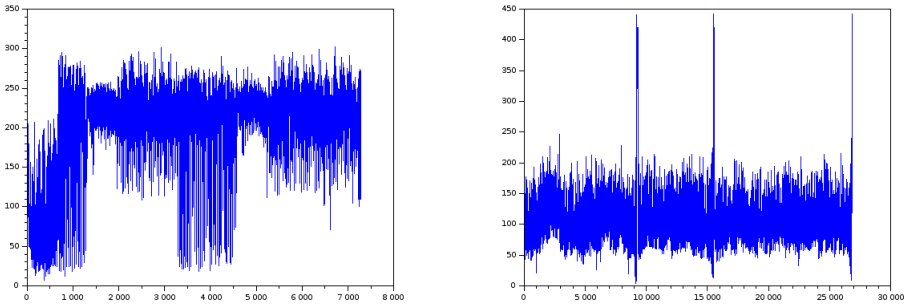

FIGURE 1 – Rolloff (quantité de basses) d'un morceau d'electro (à gauche) et d'un morceau de classique (à droite) au cours du temps

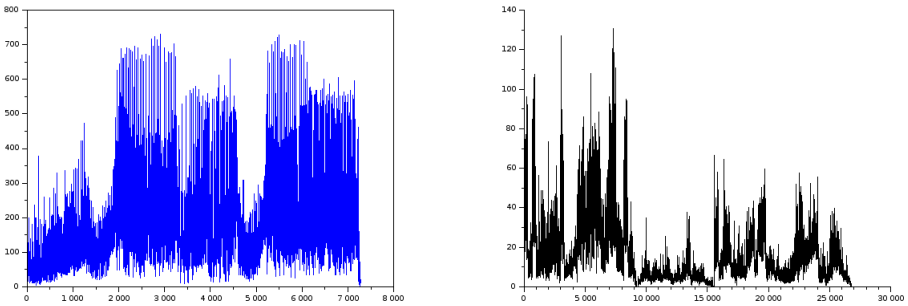

FIGURE 2 – Flux (agitation) d'un morceau d'electro (à gauche) et d'un morceau de classique (à droite) au cours du temps

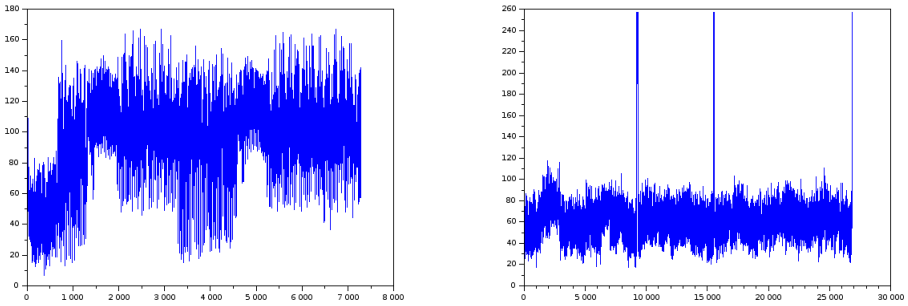

FIGURE 3 – Centroide (brillance) d'un morceau d'electro (à gauche) et d'un morceau de classique (à droite) au cours du temps

2.3 Résultats

Voici les résultats que nous avons obtenus pour 4 morceaux. 2 morceaux de musique classique (Vivaldi, Haendel) et 2 morceaux de musique électronique (AdhesiveWombat, Vexento).

Titre	Centroid	Rolloff	Flux	ZeroCross	LowEnergy
Pixel Party	92.3	195.3	173.2	81.3	52%
8-bit Adventure	94.2	205.6	108.9	80.7	48%
4 saisons, Printemps	59.2	98.8	15.7	58.0	62%
Water Music 1 Haendel	55.7	90.7	23.4	54.6	61%

On peut voir de nettes différences entre les chiffres des deux genres musicaux. On voit également qu'au sein d'un genre musical, les chiffres se ressemblent.

3 Régression logistique

3.1 Principe

N'ayant pas énormément de temps pour analyser beaucoup de fichiers audio, nous avons décidé de faire notre régression logistique sur les 4 morceaux qui ont été analysés ci-dessus. On définit avec les nombres obtenus un espace à 5 dimensions auquel on ajoute une dimension pour des scores arbitraires qu'on attribue à chaque genre.

À l'aide du tableau ci-dessus, on crée une matrice X contenant les résultats obtenus. On décide d'assigner aux morceaux de classique le score 1, et aux morceaux d'électro le score 0. Ce qui nous donne le vecteur $p = (1, 1, 0, 0)$.

Il s'agit maintenant de trouver un vecteur a et un scalaire b vérifiant : $log(\frac{p}{1-p}) = Xa + b$.

On fait cela à l'aide de la fonction *glm* de R (cf Annexe 2).

Et on obtient :

$a = (-2.22428, 0.41626, -0.08565, 0.59464, 1.57771)$ et $b = -15.78057$.

3.2 Exploitation

On décide pour valider notre algorithme d'analyser pour 4 morceaux différents de ceux que nous avons utilisé pour créer notre modèle. On analyse ici dix secondes de musique, et non pas l'intégralité des morceaux comme précédemment.

Les morceaux utilisés sont :

— 2e mouvement de la Sonate au clair de lune de Beethoven
— une sonate de Chopin
— Force - Alan Walker
— Unity - TheFatRat

On choisit les dix secondes étudiées arbitrairement dans les morceaux (ici on a choisi de les prendre au milieu du morceau). On fait la même analyse que précédemment, on obtient donc 5 chiffres représentant ces dix secondes, qu'on stocke dans un vecteur x, puis qu'on remplace dans $y = a * x + b$. On a ensuite $p = \frac{e^y}{1+e^y}$. Ce qui nous donne p, la probabilité qu'un morceau soit de la musique classique.

3.3 Résultats

Quand y est proche de 1, on dit que le morceau est "classique", quand y est proche de 0, on dit que le morceau est "electro". Notre algorithme a réussi a trouver le genre des 4 morceaux énoncés ci-dessus.

4 Annexes

4.1 Programme d'analyse

```
//validation set
fichier = "/home/blacky/Documents/2A/CECA/progs/02 - Valse - Op.34 No.1 - in Ab.wav"
//fichier = "/home/blacky/Documents/2A/CECA/progs/TheFatRat - Unity.wav"
//fichier = "/home/blacky/Documents/2A/CECA/progs/Alan Walker - Force [NCS Release].wav"
//fichier = "/home/blacky/Documents/2A/CECA/progs/Beethoven - Moonlight Sonata In C
    Sharp Minor - Allegretto.wav"

//training set
//fichier = "/home/blacky/Documents/2A/CECA/progs/Vexento - Pixel Party.wav"
//fichier = "/home/blacky/Documents/2A/CECA/progs/Antonio Vivaldi - The Four Seasons -
    Spring.wav"
//fichier = "/home/blacky/Documents/2A/CECA/progs/AdhesiveWombat - 8 Bit Adventure.wav"
//fichier = "/home/blacky/Documents/2A/CECA/progs/01 Water Music Suites Nos 1 - 3 for
    orchestra, HWV 348 - 350.wav"
stacksize('max')
disp(wavread(fichier, "info"))
y = wavread(fichier)

i=1
[x,Nsamples] = size(y)
pouic = Nsamples;
while y(1,i) == 0
    i = i +1
end
j = Nsamples
while y(1,j) == 0
    j = j -1
end

y = y(:,i:j)

[x,Nvals] = size(y)

//A COMMENTER POUR ANALYSER TOUT LE MORCEAU
start = floor(Nvals/2)
Ns = 400
samplesize = 1024
y = y(:,start:(start + Ns * samplesize))


freqs = 1:(samplesize/2)

[x,Nvals] = size(y)

Nsamples = floor(Nvals / samplesize)

Centroid = 0



//centroid
Centroid = 0
for i = 1:Nsamples
    ff = abs(fft(y(1,(1+ (i-1)*samplesize):(samplesize +(i-1)*samplesize))))
    s = sum(ff(1:(samplesize/2 )))
    if s ~= 0 then
        Centroid(i) = sum(freqs .* ff(1:(samplesize/2 ))) / sum(ff(1:(samplesize/2 )))
    else
        Centroid(i) = 0
    end

end

//plot2d(Centroid)
disp("Mean-Centroid : ")
```

```
mc = mean(Centroid)
disp(mc)

//rolloff
R=0
R(1) = 1
for i = 1:Nsamples
ff = abs(fft(y(1,(1+ (i-1)*samplesize):(samplesize +(i-1)*samplesize)),-1))
droite = 0.85 * sum(ff(1:(samplesize/2)))
acc = sum(ff(1:R(i)))
if acc < droite then
    while(acc < droite)
        if R(i) > samplesize/2 then
            break
        end
        if R(i) == 0 then
            R(i) = 1
        end
        acc = acc + ff(R(i))

        R(i) = R(i) + 1

    end
else
    while(acc > droite)
        if R(i) < 1 then
            break
        end
        acc = acc - ff(R(i))

        R(i) = R(i) - 1

    end
end

R(i+1) = R(i)
//clf()
//plot2d(ff)
//sleep(250)
//disp(R(i))
end
clf()
plot2d(R)
disp("Mean-Rolloff␣:␣")
mr = mean(R)
disp(mr)

i=1
prevff = abs(fft(y(1,(1+ (i-1)*samplesize):(samplesize +(i-1)*samplesize))))
Flux = 0
//FLUX
for i = 2:Nsamples

    ff = abs(fft(y(1,(1+ (i-1)*samplesize):(samplesize +(i-1)*samplesize))))

    Flux(i-1) = norm(ff - prevff)
    prevff = ff
end

disp("Mean-Flux␣:␣")
clf()
plot2d(Flux)
mf = mean(Flux)
disp(mf)


//zerocrossings
signal = y(1,1:(Nvals - 1))
shift = y(1,2:(Nvals))
```

```
zerocross = sum(((signal > 0) & (shift < 0 )) | ((signal < 0) & (shift > 0 ))) /
    Nsamples

disp("ZeroCross␣:␣")
disp(zerocross)

//lowenergy

Energies = 0

for i = 1:Nsamples
    ff = abs(fft(y(1,(1+ (i-1)*samplesize):(samplesize +(i-1)*samplesize))))
    Energies(i) = norm(ff)
end

disp("LowEnergy␣:␣")
lowenergy = 100 * (sum(Energies < mean(Energies)) / Nsamples)
disp(lowenergy)

valeurs = [mc mr mf zerocross lowenergy]

predict(valeurs)
```

4.2 Regression logistique

```
library("MASS")

X <- matrix( c(59.524279, 98.790711, 15.664274,  58.049465,   62.39803,1,
               55.704163, 90.675298, 23.371763,  54.55436,  60.88922,1,
               92.329056, 195.34717, 173.21853,  81.29567, 51.82131, 0,
               94.201679, 205.56836, 108.96227,  80.67663,  48.50983, 0,
               49.608974, 81.271908, 11.700325, 25.578547, 65.064499, 1,
               93.077609, 195.46014, 88.520738, 80.744751, 48.456701, 0),
             nrow = 6, ncol = 6, byrow = TRUE)

dat = data.frame(X)

model = glm(X6 ~ X1 + X2 + X3 + X4 + X5, family = binomial(logit), data=dat)
```

4.3 Fonction de prédiction

```
// prediction
function res = predict(X)

a = [-2.22428 ,0.41626 ,-0.08565 ,0.59464 ,1.57771];

b = -15.78057 ;

res = a * X' + b;
res = (exp(res)/(1+exp(res)))
disp(res)

if res < 0.5 then
    disp("electro")
else
    disp("classique")
end
endfunction
```